# 系统节拍定时器

**系统节拍定时器**

ARM Cortex-M4处理器，提供了一个系统节拍定时器:SysTick

* 24位递减计数，使用系统时钟
* 用于实时操作系统的节拍计数器
* 用于精确延时
* 用于测量一段程序的执行时间

系统节拍定时器由三个寄存器控制

节拍控制和状态寄存器STCTRL：配置时钟、使能、使能中断、查看状态

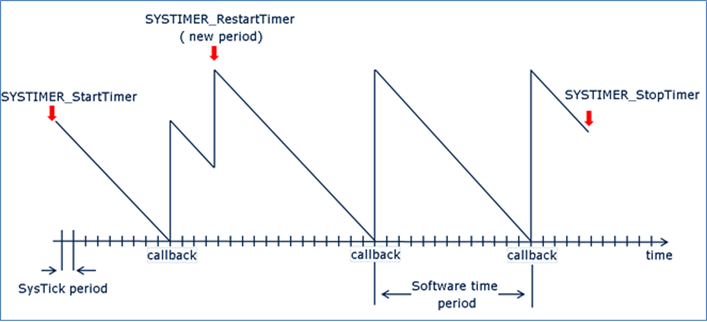
节拍计数初值寄存器STRELOAD: 计数器从这个值开始计数，向下计数，计到零后，又重新从这个值开始计数

节拍当前值寄存器STCURRENT：计数器的当前值

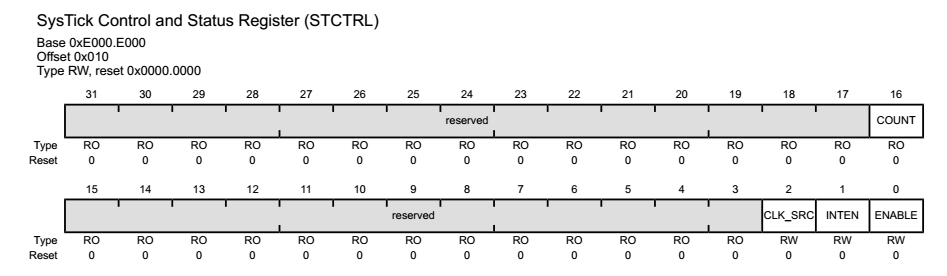
**系统节拍定时器的工作原理**

使能后，系统节拍定时器的计数器开始向下递减计数，每过一个系统时钟周期，计数值减一。

读取STCURRENT寄存器可以获知当前的计数值。



**控制和状态寄存器STCTRL**



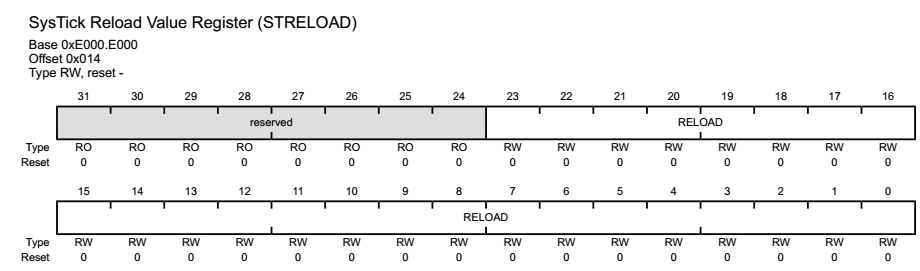
COUNT：计数器计数到零后，COUNT位置1。读取该寄存器，使COUNT清零。写STCURRENT寄存器，也可以使COUNT清零。

CLK\_SRC：时钟源选择。0：PIOSC/4。1：系统时钟

INTEN：中断使能控制。0：关闭SysTick中断。1：计数到0后，产生SysTick中断到NVIC。

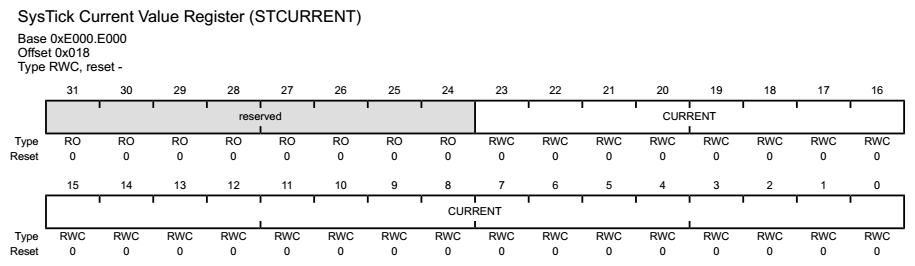
ENABLE：计数器使能控制。0：关闭计数器。1：开启计数器，计数器从STRELOAD开始计数。

**节拍计数初值寄存器STRELOAD**



RELOAD ：24位计数初值，可以是1到0x00FF FFFF之间的任何数。如果写0的话，会关闭计数器，使计数器不工作。但不会产生中断。计数器从1数到0的过程中，产生中断。

节拍当前值寄存器STCURRENT



CURRENT 24位当前计数值

写入任何数据，都会使CURRENT清零，并清除STCTRL的COUNT位，然后从24位计数初值STRELOAD重新开始计数。

# 系统节拍定时器的操作方法

**系统节拍定时器的操作方法**

与节拍定时器有关的函数和宏定义在driverlib/systick.h和inc/hw\_nvic.h中，因此用到节拍定时器后，要包含这两个文件。

**系统节拍定时器的使用步骤：**

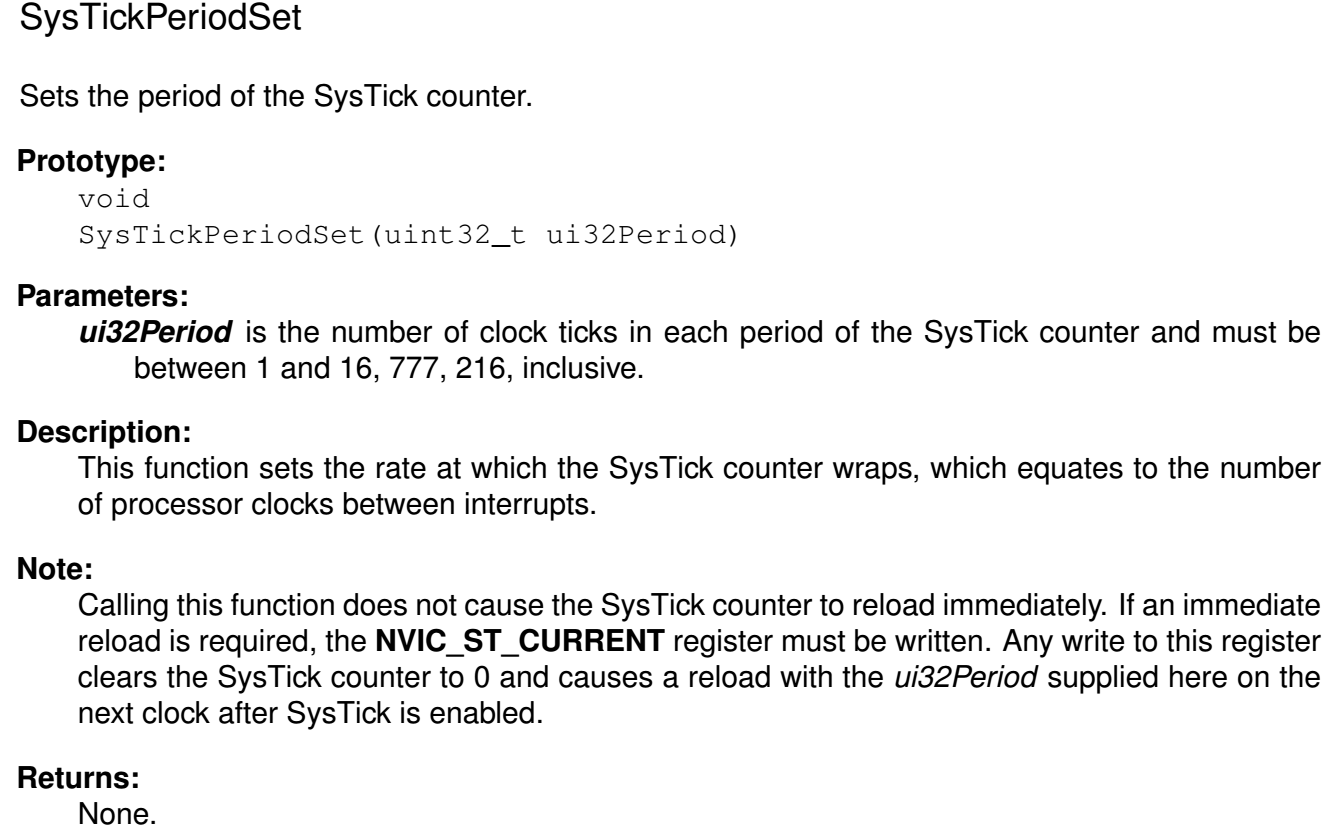
1 设置计数初值STRELOAD

2 向STCURRENT写任意值，使STCURRENT清零

3 设置STCTRL并开启计数

1. **设置计数初值STRELOAD**

库函数提供了SysTickPeriodSet函数，用于设置计数初始值。



SysTickPeriodSet函数在TivaWare\_C\_Series-2.1.4.178\driverlib\systick.c中定义，已经包含着库函数中

void

SysTickPeriodSet(uint32\_t ui32Period)

{

ASSERT((ui32Period > 0) && (ui32Period <= 16777216));

// Set the period of the SysTick counter.

HWREG(NVIC\_ST\_RELOAD) = ui32Period - 1;

}

由于COUNT置位和中断请求都发生在从1数到0的时刻，而且计数第一个周期是从0变为STRELOAD，因此设置在STRELOAD中的计数值应该-1.

也可以直接操作NVIC\_ST\_RELOAD，而不必调用SysTickPeriodSet函数。

1. **向STCURRENT写任意值，使STCURRENT清零**

没有专用的函数完成这个功能，需要直接操作**STCURRENT寄存器。**

HWREG(NVIC\_ST\_CURRENT)=0;

1. **设置STCTRL并开启计数**

**库函数提供了下面四个函数，控制计数器的使能和计数器中断的使能。**

void SysTickDisable (void)

void SysTickEnable (void)

void SysTickIntDisable (void)

void SysTickIntEnable (void)

SysTickEnable 设置计数器时钟为系统时钟，并且开启计数器

SysTickEnable 函数在TivaWare\_C\_Series-2.1.4.178\driverlib\systick.c中定义，已经包含着库函数中

void

SysTickEnable(void)

{

HWREG(NVIC\_ST\_CTRL) |= NVIC\_ST\_CTRL\_CLK\_SRC | NVIC\_ST\_CTRL\_ENABLE;

}

**SysTickDisable 关闭计数器**

SysTickDisable 函数在TivaWare\_C\_Series-2.1.4.178\driverlib\systick.c中定义，已经包含着库函数中

void

SysTickDisable(void)

{

HWREG(NVIC\_ST\_CTRL) &= ~(NVIC\_ST\_CTRL\_ENABLE);

}

**SysTickIntEnable 开启计数器中断**

SysTickIntEnable 函数在TivaWare\_C\_Series-2.1.4.178\driverlib\systick.c中定义，已经包含着库函数中

void

SysTickIntEnable(void)

{

HWREG(NVIC\_ST\_CTRL) |= NVIC\_ST\_CTRL\_INTEN;

}

**SysTickIntDisable 关闭计数器中断**

SysTickIntDisable 函数在TivaWare\_C\_Series-2.1.4.178\driverlib\systick.c中定义，已经包含着库函数中

void

SysTickIntDisable(void)

{

HWREG(NVIC\_ST\_CTRL) &= ~(NVIC\_ST\_CTRL\_INTEN);

}

可以看出，这几个函数，是通过操作STCTRL对应的位，来实现相应的功能的。因此，也可以直接对NVIC\_ST\_CTRL进行操作。

**获取计数值SysTickValueGet**

SysTickValueGet函数在TivaWare\_C\_Series-2.1.4.178\driverlib\systick.c中定义，已经包含着库函数中

uint32\_t

SysTickValueGet(void)

{

//

// Return the current value of the SysTick counter.

//

return(HWREG(NVIC\_ST\_CURRENT));

}

**注册中断服务函数SysTickIntRegister**

SysTickIntRegister函数在TivaWare\_C\_Series-2.1.4.178\driverlib\systick.c中定义，已经包含着库函数中

void

SysTickIntRegister(void (\*pfnHandler)(void))

{

IntRegister(FAULT\_SYSTICK, pfnHandler);

// Enable the SysTick interrupt.

HWREG(NVIC\_ST\_CTRL) |= NVIC\_ST\_CTRL\_INTEN;

}

SysTickIntRegister调用IntRegister来注册中断服务函数，然后开启节拍计数器的中断。

IntRegister将向量表移动到SRAM中，然后修改向量表，注册中断服务函数。

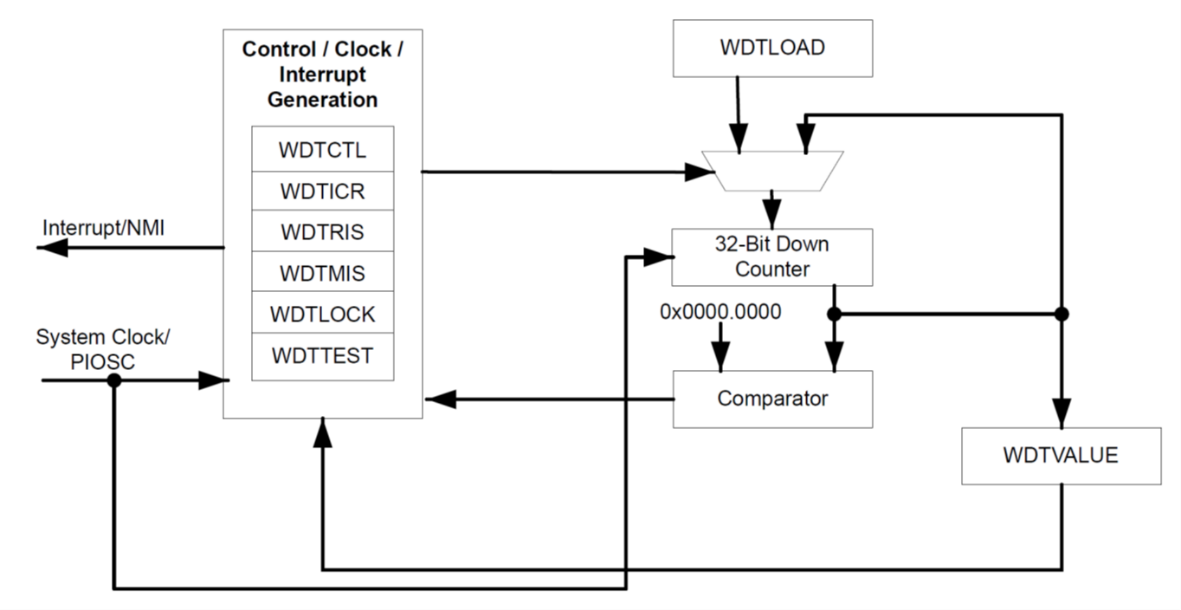
# 看门狗计数器的工作原理

**看门狗计数器Watchdog Timer：**

用于在系统产生错误或者程序出现故障、失去响应时，使系统复位。

**看门狗计数器的工作原理：**

* 看门狗计数器设置一定的计时时间。
* 看门狗使能后，计数器开始向下计数。
* 当计时时间到后，则触发系统复位。
* 如果在定时时间到达之前，进行喂狗（计数器重装）动作，就不会引起系统复位。



使能后，计数器从WDTLOAD开始向下计数。

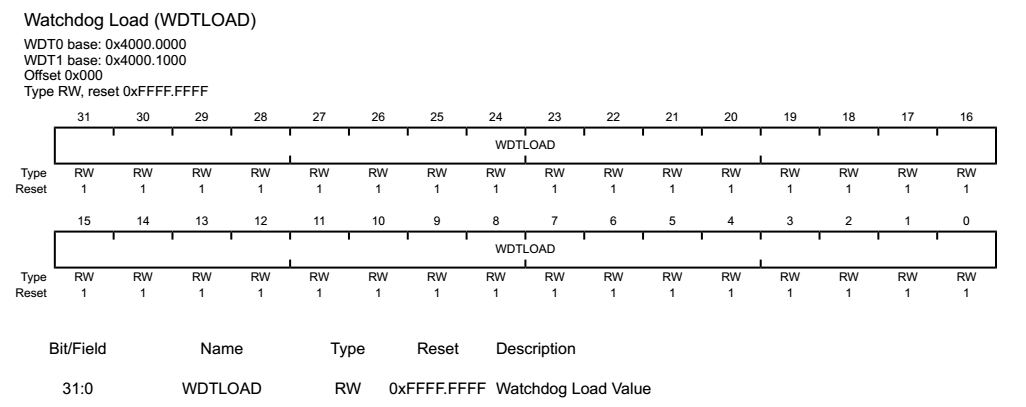
当看门狗定时器第一次递减到0时，产生一个超时信号，触发中断。

如果第一次超时的中断状态没有被清除，计数器第二次递减到零后，系统复位。

**特点：**

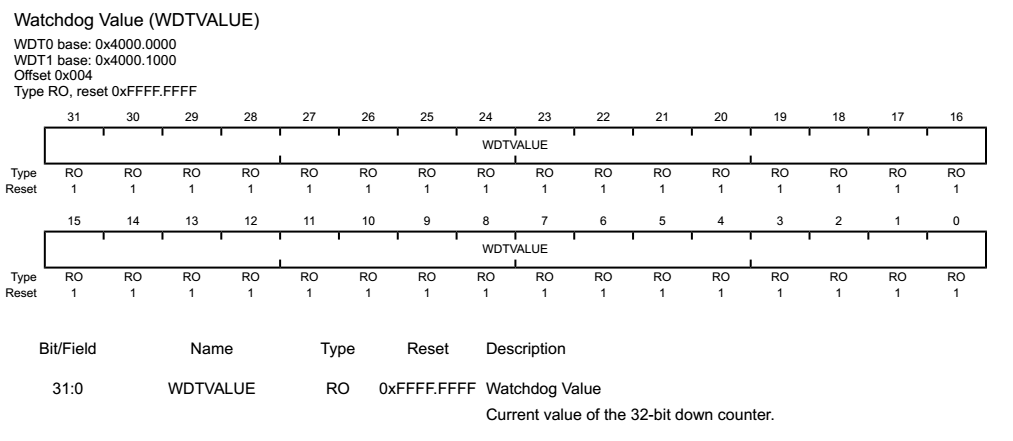
* 32位递减计数器，计数周期可编程
* 可设置不同的时钟源，带使能控制
* 可编程中断
* 内部寄存器保护
* 可使能/禁止产生复位信号
* 调试时可以暂停

**计数初值寄存器WDTLOAD**



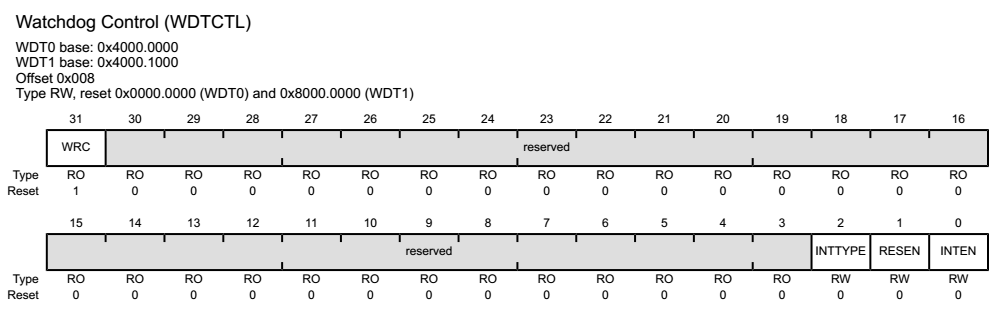
32位计数初值。

**当前计数值寄存器 WDTVALUE**



32位当前计数值

看门狗控制寄存器 WDTCTL

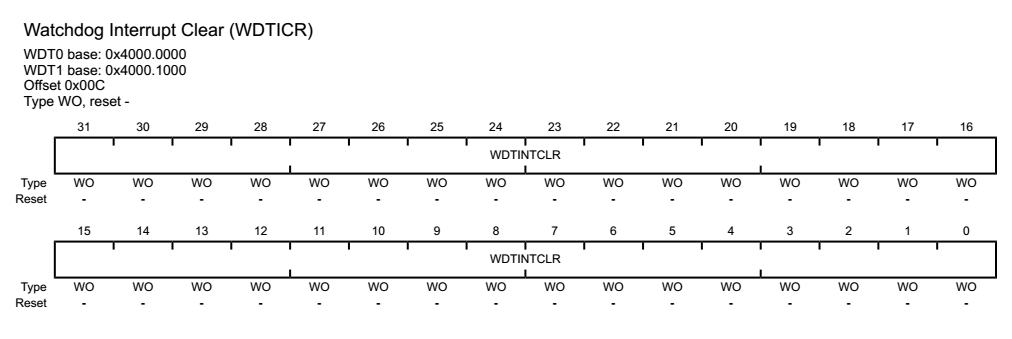


INTTYPE： 看门狗产生的中断类型。0：普通可屏蔽中断。1：非可屏蔽中断NMI

RESEN：重置使能。0：看门狗不能重启系统。1：看门口可以重启系统。写1后，开门狗定时器启动。

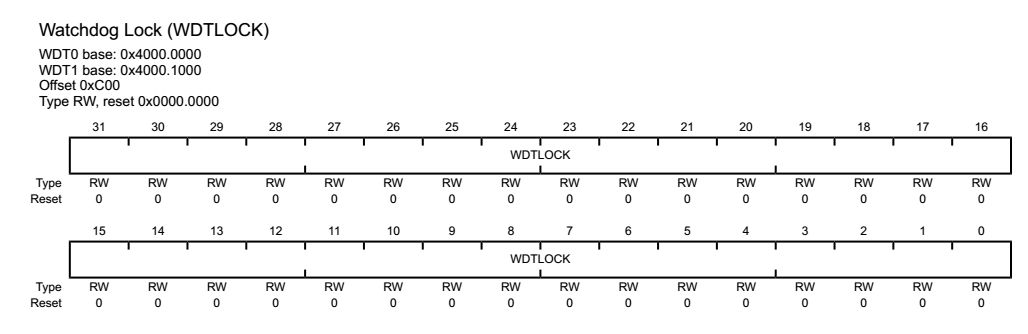
INTEN：中断使能。0：关闭中断。1：启动中断，一旦写1，除非重启系统，该位的值都不会再改变

中断清除寄存器 WDTICR



向该寄存器写任意数值，都可清除看门狗中断，并且重置看门狗计数值为WDTLOAD

看门狗锁定寄存器 WDTLOCK



向该寄存器中写入0x1ACC E551，可以解锁看门狗寄存器组，使其可编辑。写入其他数值，锁定看门狗寄存器组，是其不可编辑。

如果看门狗寄存器组已经锁定，读取该寄存器返回1。如果未锁定，返回0。

# 看门狗计数器的使用

**看门狗定时器的使用**

与看门狗有关的库函数和宏定义位于driverlib/watchdog.h中，使用看门狗模块，需要包含该文件。

1. **在系统控制模块中开启看门狗模块的时钟**

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_WDOG0); // #define SYSCTL\_PERIPH\_WDOG0 0xf0000000 // Watchdog 0

SysCtlPeripheralEnable函数定义如下：

void

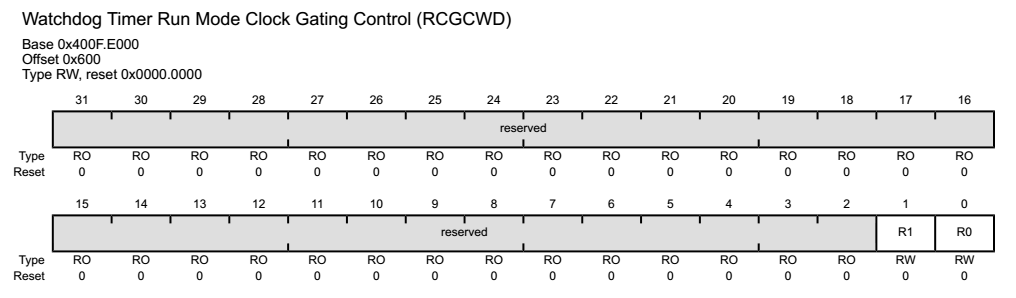
SysCtlPeripheralEnable(uint32\_t ui32Peripheral)

{

HWREGBITW(SYSCTL\_RCGCBASE + ((ui32Peripheral & 0xff00) >> 8),

ui32Peripheral & 0xff) = 1;

} //#define SYSCTL\_RCGCBASE 0x400fe600



1. **解锁看门狗寄存器组**

**if**(**WatchdogLockState**(WATCHDOG0\_BASE) == true){

**WatchdogUnlock**(WATCHDOG0\_BASE);

}

库函数提供了WatchdogLockState查看看门狗锁的状态，并提供WatchdogUnlock解锁看门狗寄存器组。

WatchdogLockState和WatchdogUnlock在TivaWare\_C\_Series-2.1.4.178\driverlib\watchdog.c中定义，已经包含着库函数中

bool

WatchdogLockState(uint32\_t ui32Base)

{

return((HWREG(ui32Base + WDT\_O\_LOCK) == WDT\_LOCK\_LOCKED) ? true : false);

}

其中：#define WDT\_O\_LOCK 0x00000C00 // Watchdog Lock

#define WDT\_LOCK\_UNLOCKED 0x00000000 // Unlocked

#define WDT\_LOCK\_LOCKED 0x00000001 // Locked

WatchdogLockState函数读取WDTLOCK寄存器，查看看门狗的状态

void

WatchdogUnlock(uint32\_t ui32Base)

{

HWREG(ui32Base + WDT\_O\_LOCK) = WDT\_LOCK\_UNLOCK;

}

其中 #define WDT\_LOCK\_UNLOCK 0x1ACCE551 // Unlocks the watchdog timer

WatchdogUnlock 函数向WDTLOCK寄存器写0x1ACCE551 ，来解锁看门狗寄存器组

1. **设置看门狗计数器的计数初值**

**WatchdogReloadSet**(WATCHDOG0\_BASE, ui32SysClock\*5);

调用WatchdogReloadSet函数，设置计数初始值。WatchdogReloadSet函数在TivaWare\_C\_Series-2.1.4.178\driverlib\watchdog.c中定义，已经包含着库函数中，他的实现形式如下。

void

WatchdogReloadSet(uint32\_t ui32Base, uint32\_t ui32LoadVal)

{

HWREG(ui32Base + WDT\_O\_LOAD) = ui32LoadVal;

}

WatchdogReloadSet函数通过写WDTLOAD寄存器，设置初始值。

练习：以上代码中，看门狗的超时时间是多长？

1. **注册中断服务函数**

**WatchdogIntRegister**(WATCHDOG0\_BASE,WatchdogIntHandler);

该函数将WatchdogIntHandler注册为中断服务函数，然后使NVIC接收看门狗模块的中断。WatchdogIntRegister在TivaWare\_C\_Series-2.1.4.178\driverlib\watchdog.c中定义，已经包含着库函数中，他的实现方式如下：

void

WatchdogIntRegister(uint32\_t ui32Base, void (\*pfnHandler)(void))

{

IntRegister(INT\_WATCHDOG\_TM4C123, pfnHandler);

IntEnable(INT\_WATCHDOG\_TM4C123);

}

1. **使能看门狗模块的中断和重启功能**

**WatchdogIntEnable**(WATCHDOG0\_BASE);

**WatchdogResetEnable**(WATCHDOG0\_BASE);

这两个函数在TivaWare\_C\_Series-2.1.4.178\driverlib\watchdog.c中定义，已经包含着库函数中，他们的实现方式如下：

void

WatchdogIntEnable(uint32\_t ui32Base)

{

HWREG(ui32Base + WDT\_O\_CTL) |= WDT\_CTL\_INTEN;

}

void

WatchdogResetEnable(uint32\_t ui32Base)

{

HWREG(ui32Base + WDT\_O\_CTL) |= WDT\_CTL\_RESEN;

}

这两个函数操作WDCTRL寄存器的INTEN和RESEN两个位，这两个位中的任意一个被写1后，看门狗都会自动开始运行。

如果直接使用寄存器操作，可以用一句代码完成上述两个功能，请写出代码：

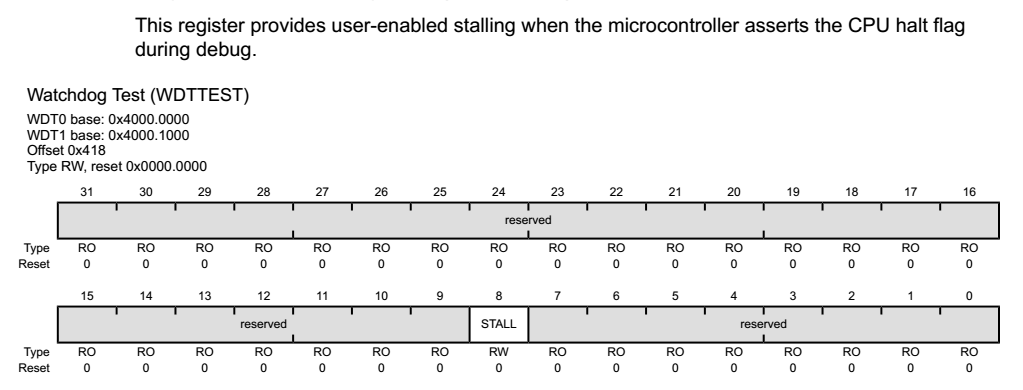
HWREG(WATCHDOG0\_BASE + WDT\_O\_CTL) |= (WDT\_CTL\_INTEN | WDT\_CTL\_RESEN);

或

HWREG(WATCHDOG0\_BASE + WDT\_O\_CTL) |= 0x03;

使能调试功能

调试器暂停CPU的时候，看门狗还会继续工作，这会导致系统意外重启。WDTTEST寄存器可以在这种情况下，暂停看门狗的计数。



**WatchdogStallEnable**(WATCHDOG0\_BASE);

WatchdogStallEnable在TivaWare\_C\_Series-2.1.4.178\driverlib\watchdog.c中定义，已经包含着库函数中，他的实现方式如下：

void

WatchdogStallEnable(uint32\_t ui32Base)

{

HWREG(ui32Base + WDT\_O\_TEST) |= WDT\_TEST\_STALL;

}

即操作WDTEST寄存器，将STALL位置1。

1. **锁定看门狗寄存器组**

**WatchdogLock**(WATCHDOG0\_BASE);

WatchdogLock将看门狗寄存器锁定，不能随意更改，WatchdogLock在TivaWare\_C\_Series-2.1.4.178\driverlib\watchdog.c中定义，已经包含着库函数中，他的实现方式如下：

void

WatchdogLock(uint32\_t ui32Base)

{

HWREG(ui32Base + WDT\_O\_LOCK) = WDT\_LOCK\_LOCKED; //#define WDT\_LOCK\_LOCKED 0x00000001 // Locked

}

WatchdogLock向WDTLOCK寄存器中写1，锁定寄存器组

1. **在无限循环中喂狗，确保程序没有死机**

**while**(1)

{

**WatchdogIntClear**(WATCHDOG0\_BASE);

//do something

…………

WatchdogIntClear函数在TivaWare\_C\_Series-2.1.4.178\driverlib\watchdog.c中定义，已经包含着库函数中，他的实现方式如下：

void

WatchdogIntClear(uint32\_t ui32Base)

{

HWREG(ui32Base + WDT\_O\_ICR) = WDT\_RIS\_WDTRIS; //

}

其中

//#define WDT\_RIS\_WDTRIS 0x00000001 // Watchdog Raw Interrupt Status

向WTICR寄存器中，写任意数值，都可以使看门狗计数器重新计数。

1. **编写中断服务函数，在程序没有及时喂狗的情况下，做特殊处理**

**void** **WatchdogIntHandler**(**void**)

{

**WatchdogIntClear**(WATCHDOG0\_BASE);

}